Package 'GHAgricProductivityLab'

November 11, 2025

Type Package

Title Agricultural Productivity in Ghana

Version 0.0.0.9000

Author Francis Tsiboe [aut, cre] (<https://orcid.org/0000-0001-5984-1072>)

Maintainer Francis Tsiboe <ftsiboe@hotmail.com>

Contributor -

Reviewer -

Creator Francis Tsiboe

Description Provides tools and datasets for investigating the drivers of agricultural production shortfalls in Ghana.
It compiles research studies that examine farmer-specific and institutional factors to assess whether inefficiencies arise from technical inefficiency, technology gaps, or both. The package offers empirical evidence to inform policy discussions and interventions aimed at improving agricultural productivity, particularly in contexts with limited access to modern technologies.

License GPL-3 + file LICENSE

URL https://github.com/ftsiboe/GHAgricProductivityLab

BugReports https://github.com/ftsiboe/GHAgricProductivityLab/issues

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

VignetteBuilder knitr

Depends R (>= 4.1.0)

Imports MatchIt, data.table, haven, stats

Remotes github::dylan-turner25/rfcip

Suggests dplyr, knitr, crayon, tidyr, withr, rmarkdown, cobalt, dplyr,doBy, stringr, testthat (>= 3.0.0)

LazyData true

Cite-us If you find it useful, please consider staring the repository and citing the following studies
- Tsiboe, F. and Turner, D. (2025). ``Incorporating buy-up price loss coverage into the United States farm safety net." Applied Economic Perspectives and Policy.
- Tsiboe, F., et al. (2025). ``Risk reduction impacts of crop insurance in the United States."

Applied Economic Perspectives and Policy.
- Gaku, S. and Tsiboe, F. (2024). Evaluation of alternative farm safety net program combination strategies. Agricultural Finance Review.

Contents

`covariate_balance` *Compute Covariate Balance Summaries Across Matching Specs*

Description

Compute Covariate Balance Summaries Across Matching Specs

Usage

```
covariate_balance(match_specifications, matching_output_directory)
```

Arguments

`match_specifications`
: data.frame/data.table with at least columns: `boot`, `ARRAY`, and the spec fields stored in RDS (e.g., method, distance, link).

`matching_output_directory`
: Directory containing RDS files named as "0001.rds", "0002.rds", etc.

Details

Reads each matching result RDS (expected to contain `m.out` and `match_specifications`), extracts balance via `cobalt::bal.tab`, reshapes, computes a composite balance $rate = mean((Diff - 0)^2, (KS - 0)^2, (V_Ratio - 1)^2)$, and averages by `ARRAY`, `method`, `distance`, `link`, `sample`.

Value

A list with:

`rate`
: data.frame of mean balance metrics by spec (Adj sample only) with a composite `rate`.

`bal_tab`
: long-format balance table per covariate/stat/sample/spec.

`draw_matched_samples` *Draw matched samples (stratified bootstrap + matching)*

Description

Performs stratified resampling at the `Surveyx, EaId` level (excluding EaIds listed for the current bootstrap `ID`) and computes adjusted sampling weights used for matching. Then fits a matching model per `match_specifications[i, ]` using **MatchIt**, with exact matching on `Emch` and distance/link from `match_specifications`.

Usage

```
draw_matched_samples(
  i,
  data,
  match_variables_exact,
  match_variables_scaler,
  match_variables_factor,
  match_specifications,
  sample_draw_list,
  verbose = FALSE
)
```

Arguments

`i`	Integer index selecting the row of `match_specifications` to use.
`data`	A data.frame/data.table containing at least: `Surveyx, EaId, HhId, Mid, UID, Weight, Treat`, plus columns named in `Emch, Scle, Fixd`.
`match_variables_exact`	A character vector of variable names to be included in the exact match component (e.g., `"gender"`, `"region"`).
`match_variables_scaler`	A character vector of continuous or scalar variable names to include as covariates in the general match formula (e.g., `"age"`, `"income"`).
`match_variables_factor`	A character vector of factor variable names to be included as dummy-coded categorical covariates in the general match formula.
`match_specifications`	Data frame of matching specifications with columns `boot, method, distance`, and optionally `link`.
`sample_draw_list`	Data frame where column `ID` identifies the bootstrap draw, and each remaining column corresponds to a `Survey` containing the sampled `EaId`.
`verbose`	Logical; if `TRUE`, prints the chosen matching method and timing. Default `FALSE`.

Details

Adjusted weights are computed as $pWeight = Weight \times (alloc/allocj)$, where `alloc` is the pre-exclusion sum of `Weight` by `Surveyx, EaId` and `allocj` is the post-exclusion sum.

Exact matching is performed on variables in Emch. The distance model formula is constructed as Treat ~ Scle + Fixd. When distance == "glm", m.order = "largest" is used; otherwise "closest".

Value

A list with:

match_specifications The selected row from match_specifications.

m.out The matchit object.

md Matched data: Surveyx, EaId, HhId, Mid, UID, weights, pWeight.

df The analysis data with adjusted weights: Surveyx, EaId, HhId, Mid, UID, pWeight.

---

draw_match_sample_specifications

Draw / Match Sample Specifications

Description

Generates (i) a draw list by sampling EaId within each unique Survey group and (ii) a grid of matching specifications for each bootstrap draw.

Usage

```
draw_match_sample_specifications(drawN, data, myseed = 3242025)
```

Arguments

drawN	Integer. The number of draws to perform per Survey.
data	A data.frame or data.table containing at least the columns Survey and EaId.
myseed	Integer. Seed for random number generation (default 03242025).

Details

Draw list: For each unique value of Survey, the function samples drawN EaId values with replacement and prepends a 0 row (ID = 0) for the baseline. The result is then spread to wide format with one column per Survey.

Matching specs: For each draw ID, creates a set of matching model specifications that include:

- Nearest neighbor with distances: "euclidean", "scaled_euclidean", "mahalanobis", "robust_mahalanobis".
- Nearest neighbor with distance "glm" and links: "logit", "probit", "cloglog", "cauchit".

An ARRAY index is added for convenience.

Value

A list with two elements:

m.specs A data.frame of matching specifications with columns boot, method, distance, link, ARRAY.

drawlist A data.frame in wide format where each Survey is a column and rows correspond to draw ID (0:drawN).

Note

Requires that `data` contain `Survey` and `EaId`. `tidyr::spread()` is used for wide reshaping (consider `tidyr::pivot_wider()` in new code).

`functional_forms` *Generate Functional and Distribution Forms*

Description

Creates symbolic production function strings for stochastic frontier analysis (SFA), including Cobb-Douglas (CD) and Translog (TL) specifications, plus a set of candidate distributions for the inefficiency term.

Usage

```
functional_forms(nX = 5, trend = FALSE)
```

Arguments

`nX`	Integer. Number of input variables. Default is 5.
`trend`	Logical. If `TRUE`, adjusts the (optional) transcendental form (`TP`) by removing the last linear input term. Default is `FALSE`.

Details

The returned `fxnforms` list includes:

- `CD`: sum of log inputs (`lnI1 + lnI2 + ...`).
- `TL`: CD terms plus second-order and pairwise interaction terms (e.g., `0.5 * lnIi^2` and `lnIi * lnIj` for `i < j`).

Additional forms (Linear, Quadratic, Generalized, Transcendental) are shown in the code as commented examples; uncomment to include them. The `trend` argument only affects `TP` when that form is enabled.

`distforms` lists common inefficiency distributions (e.g., half-normal, truncated normal, exponential, lognormal, Weibull) and whether a scaling property is assumed for each.

Value

A list with:

`fxnforms` Named character strings of functional forms.

`distforms` Named list of inefficiency distributions with a `scaling` flag.

get_crop_area_list *Extract Matching Crop Area Columns from a Dataset*

Description

Identifies and returns the column names in a dataset corresponding to crop area variables for a specified set of crops. The function looks for column names that start with "Area_" and match any of the crops provided in selected_crops.

Usage

```
get_crop_area_list(
  data,
 selected_crops = c("Beans", "Cassava", "Cocoa", "Cocoyam", "Maize", "Millet", "Okra",
    "Palm", "Peanut", "Pepper", "Plantain", "Rice", "Sorghum", "Tomatoe", "Yam")
)
```

Arguments

data	A data.frame or data.table containing crop-related variables. Column names are expected to include fields prefixed with "Area_", such as "Area_Maize".
selected_crops	A character vector specifying the crop names to filter. Defaults to a common set of crops including "Beans", "Cassava", "Cocoa", "Cocoyam", "Maize", "Millet", "Okra", "Palm", "Peanut", "Pepper", "Plantain", "Rice", "Sorghum", "Tomatoe", and "Yam".

Value

A character vector containing the names of columns in data that correspond to the specified crop area variables (e.g., "Area_Maize", "Area_Rice"). Returns an empty vector if no matching columns are found.

harmonized_data_prep *Prepare Data for Agricultural Productivity Analysis*

Description

Cleans and transforms a dataset by creating new variables, applying log transformations, converting selected variables to factors or characters, and recoding education levels. The function is designed to standardize inputs for further analysis of agricultural productivity.

Usage

```
harmonized_data_prep(data)
```

Arguments

data	A data.frame or data.table containing household- and farm-level variables such as weights, demographic information, and agricultural inputs.

Value

A cleaned and transformed `data.frame` or `data.table` with additional variables ready for analysis.

`sf_functional_forms` *Generate Functional and Distribution Forms for MSF Models*

Description

Builds algebraic strings for common production-function specifications and assembles a catalog of distributional assumptions for one-sided inefficiency terms used in stochastic frontier / multi-stage frontier (MSF) analysis.

Usage

```
sf_functional_forms(number_of_inputs = 5, include_trend = FALSE)
```

Arguments

`number_of_inputs`	Integer. Number of input variables to enumerate in the functional forms (creates symbols `I1..Ik` and `lnI1..lnIk`). Default `5`.
`include_trend`	Logical. If `TRUE`, adjusts the Transcendental production form (`TP`) by dropping the last linear input term. See *Note* regarding when `TP` is present. Default `FALSE`.

Details

Functional forms (returned as character strings):

- `CD`: Cobb-Douglas - sum of logs of inputs, e.g., `lnI1 + lnI2 + ... + lnIk`.
- `TL`: Translog - `CD` plus all second-order log terms (squares and cross-products), e.g. %
- `LN`: Linear (commented out in current code). %
- `QD`: Quadratic (commented out). %
- `GP`: Generalized (commented out). %
- `TP`: Transcendental (commented out; see Note).

Terms are generated programmatically for `k = number_of_inputs`, with: `Ii` denoting input levels and `lnIi` their logarithms.

Distributional forms (list of lists): Each entry is named and contains `list(name, scaling = <logical>)`. Included options: `hnormal`, `tnormal`, `exponential`, `tslaplace`, `genexponential`, `tnormal_scaled`, `rayleigh`, `uniform`, `gamma`, `lognormal`, `weibull`. The `scaling` flag indicates whether the Wang-Schmidt (2002) scaling property is used in that specification.

Value

A `list` with two components:

- `fxnforms`: named list of character strings for functional forms (e.g., `$CD`, `$TL`). Forms that are commented out in the implementation will not appear.
- `distforms`: named list of distribution specifications; each is a list with elements `[1]` = distribution name and `scaling` = logical.

Note

The current implementation defines CD and TL and removes NULL entries before returning. Other forms (LN, QD, GP, TP) are commented out. If you plan to set include_trend = TRUE, ensure TP is actually included in fxnforms; otherwise, modifying fxnforms$TP will not have any effect (and would error if TP is not defined).

---

sf_model_specifications

Build Model Specifications

Description

Construct a specification table for multi-stage frontier (MSF) analysis by combining alternative production-function forms and distributional assumptions with technology choices and disaggregation levels. The function first creates a pooled grid of functional forms (fxnforms) and distributions (distforms), then augments it with crop-specific and demographic splits, and finally expands technology choices.

Usage

```
sf_model_specifications(
  data,
  distforms = sf_functional_forms()$distforms,
  fxnforms = sf_functional_forms()$fxnforms,
  technology_variables,
  mainF = 2,
  mainD = 1,
 demographic_variables = c("Female", "Region", "Ecozon", "EduCat", "EduLevel", "AgeCat"),
 crop_list = c("Beans", "Cassava", "Cocoa", "Cocoyam", "Maize", "Millet", "Okra",
   "Palm", "Peanut", "Pepper", "Plantain", "Rice", "Sorghum", "Tomatoe", "Yam")
)
```

Arguments

data	data.frame or data.table. A dataset that (ideally) contains the columns named in demographic_variables. **Note:** The current implementation references a global object DATA (not data) when deriving demographic levels; ensure DATA exists with those columns or adapt your environment accordingly.
distforms	named list of distributional forms. If NULL, defaults to functional_forms()$distforms.
fxnforms	named list of functional forms. If NULL, defaults to functional_forms()$fxnforms.
technology_variables	character vector of technology variables. The first element is taken as the default/primary technology.
mainF	integer index of the preferred functional form (in fxnforms). Default is 2.
mainD	integer index of the preferred distribution (in distforms). Default is 1.
demographic_variables	character vector of column names used for demographic disaggregation. Defaults to c("Female","Region","Ecozon","EduCat","EduLevel","AgeCat").
crop_list	character vector of crop names for crop-level disaggregation. Defaults to a selection of common crops.

Details

Procedure (high level):

1. Build a pooled grid of all `fxnforms` * `distforms`.
2. Keep rows matching `mainF` and `mainD` and tag pooled specs.
3. Add crop-specific rows for `crop_list` (using the main form/dist).
4. Add demographic splits for each variable in `demographic_variables`, using unique levels in `DATA[, var]` (see note in *Warning*).
5. Set `TechVar` to the first element of `technology_variables`; if additional technology variables are provided, add pooled rows for them with the main form/dist.
6. Reorder to prioritize `mainD` and create an `nnm` flag with values `"fullset"` and `"optimal"`.
7. Remove demographic rows for the default demographic list and retain only `level == "Pooled"` in the final output.

Returned columns (final table):

- `disasg` - disaggregation variable (e.g., `"CropID"`).
- `level` - disaggregation level (e.g., `"Pooled"` or a crop/level).
- `TechVar` - selected technology variable.
- `f` - index of functional form (position in `fxnforms`).
- `d` - index of distribution (position in `distforms`).
- `nnm` - marker column (`"fullset"` / `"optimal"`).

Value

A `data.frame` (data.table-compatible) of model specifications with columns `disasg`, `level`, `TechVar`, `f`, `d`, and `nnm`.

Warning

The function calls `unique(DATA[, w])` inside a `tryCatch`; this requires a global object `DATA` containing the demographic columns. If `DATA` is absent or lacks a given variable, that split is skipped.

Assumptions

- `technology_variables` is non-empty; its first element is the default.
- `mainF` and `mainD` are valid indices within `fxnforms` and `distforms`, respectively.

study_setup *Initialize Study Environment and Directory Structure*

Description

Initialize Study Environment and Directory Structure

Usage

```
study_setup(
  seed = 1980632,
  project_name,
 local_directories = list(output = file.path("replications", project_name, "output"),
    matching = file.path("replications", project_name, "output", "matching"),
    treatment_effects = file.path("replications", project_name, "output",
  "treatment_effects"), estimations = file.path("replications", project_name, "output",
  "estimations"), figure_data = file.path("replications", project_name, "output",
  "figure_data"), figure = file.path("replications", project_name, "output", "figure"))
)
```

Arguments

seed	Numeric/integer scalar seed. Will be coerced to integer. Default: 1980632.
project_name	Length-1, non-NA character project name (required).
local_directories	Named list of character paths to create. Defaults use project_name.

Value

List with wd (directories) and seed.

treatment_effect_calculation *Compute log-linear treatment effects (ATE/ATET/ATEU) for multiple outcomes*

Description

For a given matching specification index i, this function:

1. loads the corresponding matched sample weights from disk,
2. merges them into the analysis data via UID,
3. normalizes selected inputs/outputs by plot Area,
4. fits a weighted log-linear model for each outcome with interactions between Treat and the provided matching covariate formulas,
5. transforms fitted differences to percentage effects, trims extreme values, and computes weighted ATE, ATET, and ATEU alongside model fit statistics.

Note: The function returns a long table of results (atet_scalar), not the index i.

Usage

```
treatment_effect_calculation(
  data,
  i,
  match_specifications,
  matching_output_directory,
  match_formulas
)
```

Arguments

data	A data.frame containing at least: • identifiers: UID • treatment flag: Treat (logical or 0/1) • plot size: Area (positive; used to area-normalize variables) • outcomes: HrvstKg, Area, SeedKg, HHLaborAE, HirdHr, FertKg, PestLt • any covariates referenced in match_formulas$general_match and match_formulas$exact_ma
i	Integer index selecting the row of match_specifications and the matching result file to load (zero-padded via ARRAY to "NNNN.rds").
match_specifications	A data.frame with at least column ARRAY; the i-th row is used to locate the matching output file and is also cbind-ed to the results.
matching_output_directory	Directory containing per-specification matching results named as "0001.rds", "0002.rds", etc. Each RDS should contain $md with UID and weights.
match_formulas	A list with: • general_match: a **formula** (RHS accessed via as.character(...)[3]) • exact_match: a **formula** (RHS accessed via as.character(...)[2])

Details

For each outcome in c("HrvstKg","Area","SeedKg","HHLaborAE","HirdHr","FertKg","PestLt"), fits:

```
log(outcome + eps) ~ Treat * (general_match_rhs + exact_match_rhs)
```

where eps = min(outcome[outcome>0]) * 1/100. Treatment effect is

$$TE_O LS = (e^{\hat{y}_1 - \hat{y}_0} - 1) \times 100$$

trimmed to $[-100, 100]$.

Value

A data.frame (atet_scalar) with columns from match_specifications[i,] joined to rows:

- outcome, treatment
- level in {ATE,ATET,ATEU,aic,ll,R2,N,Ft,R2a}
- est (numeric estimate)

`treatment_effect_summary` *Summarize Treatment Effect Estimates Across Matching Specifications*

Description

Aggregates and summarizes treatment effect results stored as `.rds` files in a specified output directory. The function reads each RDS file, combines all results into a single data frame, filters out invalid estimates, and computes jackknife-style summary statistics (mean, standard error, sample size, t-value, and p-value) by matching specification and outcome.

Usage

```
treatment_effect_summary(treatment_effects_output_directory)
```

Arguments

`treatment_effects_output_directory`
: Character string giving the path to the directory containing `.rds` files with treatment effect results (e.g., outputs of `treatment_effect_calculation()` saved across specifications).

Details

The function proceeds through the following steps:

1. Lists and reads all `.rds` files in `treatment_effects_output_directory`.
2. Combines them into a single long-format data frame using `data.table::rbindlist()`.
3. Removes rows with invalid estimates (`NA`, `NaN`, `Inf`, or `-Inf`).
4. Separates base (non-bootstrap) results where `boot == 0` and joins these with jackknife summary statistics computed via `doBy::summaryBy()` across method, distance, link, outcome, and level.
5. Renames summary columns to:
   - `jack_mean`: mean estimate
   - `jack_se`: standard error
   - `jack_n`: sample size (number of replicates)
   - `jack_tvl`: t-value (`jack_mean / jack_se`)
   - `jack_pvl`: two-sided p-value

Value

A `data.frame` containing summarized treatment effect estimates with columns:

- `method`, `distance`, `link`, `outcome`, `level`
- `est` (original estimate for non-bootstrap sample)
- `jack_mean`, `jack_se`, `jack_n`, `jack_tvl`, `jack_pvl`

`write_match_formulas` *Construct Matching Formulas for Exact and General Matching*

Description

Builds two types of matching formulas commonly used in propensity score or covariate matching workflows:

- An **exact match** formula specifying categorical variables to match exactly.
- A **general match** formula specifying numeric and factor covariates for distance-based matching.

The function returns both formulas as character strings that can be used in matching functions such as `MatchIt::matchit()` or `Matching::Match()`.

Usage

```
write_match_formulas(
  match_variables_exact,
  match_variables_scaler,
  match_variables_factor
)
```

Arguments

`match_variables_exact`
: A character vector of variable names to be included in the exact match component (e.g., `"gender"`, `"region"`).

`match_variables_scaler`
: A character vector of continuous or scalar variable names to include as covariates in the general match formula (e.g., `"age"`, `"income"`).

`match_variables_factor`
: A character vector of factor variable names to be included as dummy-coded categorical covariates in the general match formula.

Value

A named list with two elements:

`exact_match` A string representing the exact match formula (factor variables only).

`general_match` A string representing the general match formula, including both continuous and factor covariates on the right-hand side of `Treat ~`.

Index